● What was the bottleneck?

Subquery in WHERE clause creates two full table scans

Diagram

Description automatically generated

-> Inner hash join (student.id = `<subquery2>`.studId) (cost=414.91 rows=400) (actual

time=0.147..0.320 rows=2 loops=1)

-> Table scan on Student (cost=5.04 rows=400) (actual time=0.014..0.169 rows=400 loops=1)

-> Hash

-> Table scan on <subquery2> (cost=0.26..2.62 rows=10) (actual time=0.000..0.001 rows=2 loops=1)

-> Materialize with deduplication (cost=11.51..13.88 rows=10) (actual time=0.109..0.110 rows=2

loops=1)

-> Filter: (transcript.studId is not null) (cost=10.25 rows=10) (actual time=0.067..0.105 rows=2

loops=1)

-> Filter: (transcript.crsCode = <cache>((@v4))) (cost=10.25 rows=10) (actual

time=0.067..0.105 rows=2 loops=1)

-> Table scan on Transcript (cost=10.25 rows=100) (actual time=0.049..0.089 rows=100

loops=1)

● How did you identify it?

Using both EXPLAIN and EXPLAIN ANALYZE.

● What method you chose to resolve the bottleneck

First, I tried using an INNER JOIN rather than a subquery in WHERE clause. This improved speed only slightly.

Next, I created two indexes, one on student.id column and another on transcript.studId column. This significantly improved the outer query. However, the INNER JOIN was still utilizing a Full Table Scan due to the crsCode column.

Next, I created a third index on crsCode column on Transcript table. This significantly improved query speed for the INNER JOIN, resulting in 100% filter.

Diagram

Description automatically generated